Contenido

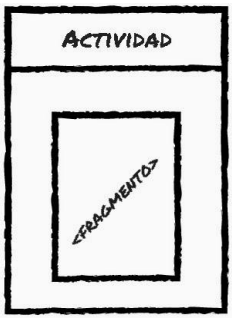
9.

Fragments

INTRODUCCIÓN

La necesidad de usar fragmentos nace con la versión 3.0 (API 11) de Android, debido a los múltiples tamaños de pantalla que estaban apareciendo en el mercado y a la capacidad de orientación de la interfaz (Landscape y Portrait). Estas características necesitaban dotar a las aplicaciones Android de la capacidad para adaptarse y responder a la interfaz de usuario sin importar el dispositivo.

Un fragmento es una sección "modular" de interfaz de usuario embebida dentro de una actividad anfitriona, el cual permite versatilidad y optimización de diseño. **Se trata de miniactividades contenidas dentro de una actividad anfitriona, manejando su propio diseño (un recurso layout propio) y ciclo de vida.**



Estas nuevas entidades permiten reusar código y ahorrar tiempo de diseño a la hora de desarrollar una aplicación. Los fragmentos facilitan el despliegue de tus aplicaciones en cualquier tipo de tamaño de pantalla y orientación.

Otra ventaja de usarlos es que permiten crear diseños de interfaces de usuario de múltiples vistas. ¿Qué quiere decir eso?, que los fragmentos son imprescindibles para generar actividades con diseños dinámicos, como por ejemplo el uso de pestañas de navegación, expand and collapse, stacking, etc.

Un fragmento se ejecuta en el contexto de una actividad, pero tiene su propio ciclo de vida y por lo general su propia interfaz de usuario, recibe sus propios eventos de entrada, y se pueden agregar o quitar mientras que la actividad exista.

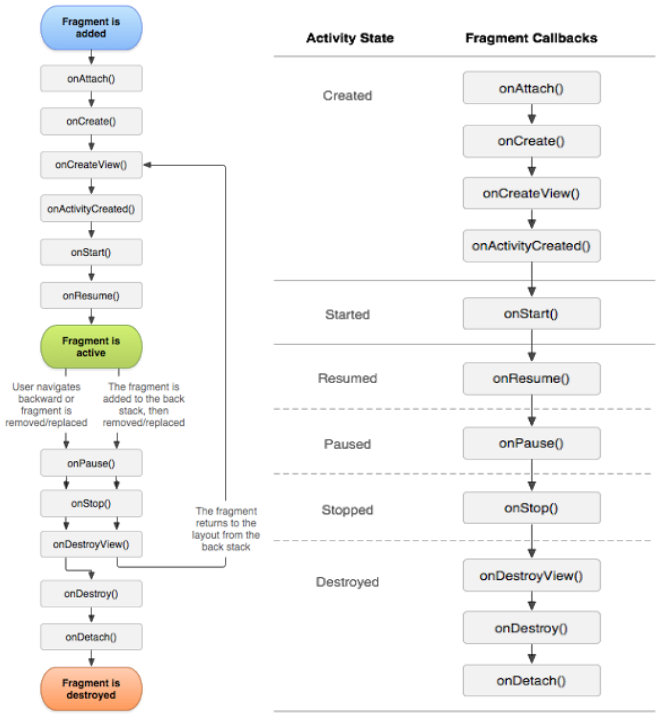
Un fragmento tiene que estar siempre integrado en una acivity, de forma que se verá afectada por el propio ciclo de vida de la activity. Por ejemplo, cuando una activity se detiene, lo hacen todos los fragmentos de la misma; cuando ésta se destruye, lo hacen también todos sus fragmentos. Sin embargo, mientras una activity está en ejecución, se puede manipular cada uno de los fragmentos incluidos en ella de forma independiente, tanto añadiéndolos como eliminándolos.

CICLO DE VIDA DE LOS FRAGMENTS

La imagen de la izquierda representa el ciclo de vida de un fragmento. Y la imagen de la derecha los estados de sus métodos.

Generalmente a la hora de crear un fragmento se deben implementar al menos los siguientes métodos:

- onCreate() El sistema llama a este método a la hora de crear el fragmento, normalmente iniciaremos los componentes esenciales del fragmento.
- onCreateView() El sistema llamara al método cuando es la hora de crear la interface de usuario o vista, es decir, asociar el layout al fragment, normalmente se devuelve la view del fragmento.
- onPause() El sistema llamara a este método en el momento que el usuario abandone el fragmento, por lo tanto es un buen momento para guardar información.



El ciclo de un fragmento tiene unos cuantos **métodos callback,** adicionales al ciclo de vida de una actividad. Averigüemos un poco sobre ellos:

onAttach: Es invocado cuando el fragmento ha sido asociado a la actividad anfitriona.

onActiviyCreated: Se ejecuta cuando la actividad anfitriona ya ha terminado la ejecución de su método ***onCreate()***.

onCreate: Este método es llamado cuando el fragmento se está creando. En el puedes inicializar todos los componentes.

onCreateView: Se llama cuando el fragmento será dibujado por primera vez en la interfaz de usuario. En este método crearemos el view que representa al fragmento para retornarlo hacia la actividad.

onStart: Se llama cuando el fragmento esta visible ante el usuario. Obviamente depende del método onStart() de la actividad.

onResume: Es ejecutado cuando el fragmento está activo e interactuando con el usuario. Esta situación depende de que la actividad anfitriona este primero en su estado Resume.

onStop: Se llama cuando un fragmento ya no es visible para el usuario debido a que la actividad anfitriona está detenida o porque dentro de la actividad se está gestionando una operación de fragmentos.

onPause: Al igual que las actividades, onPause se ejecuta cuando se detecta que el usuario dirigió el foco por fuera del fragmento.

onDestroyView: Este método es llamado cuando la jerarquía de views a la cual ha sido asociado el fragmento ha sido destruida.

onDetach: Se llama cuando el fragmento ya no está asociado a la actividad anfitriona.

SUBCLASES DE FRAGMENT

A parte de crear un Fragment directamente, Android nos ofrece la posibilidad de utilizar las siguientes subclases de Fragment:

- DialogFragment - Muestra un cuadro de dialogo flotante.
- ListFragment - Muestra una lista de elementos.
- PreferenceFragment - Muestra una lista de preferencias.

CREAR UN FRAGMENTO

Para crear un fragmento primero deberemos extender la clase "Fragment" y sobrescribir el método "onCreateView()" en el que devolveremos la vista de dicho fragmento. Vamos a ver el ejemplo y lo explicamos a continuación.

FragmentUNO.class

```
public class FragmentUNO extends Fragment
 @Override
   public View onCreateView(LayoutInflater inflater, ViewGroup container,  Bundle savedInstanceState) {
     return inflater.inflate(R.layout.fragment_uno, container, false)  }
}
```

Comentar que previamente se ha creado el layout para este fragmento y se corresponde con "fragment_uno.xml".

Al sobrescribir el método "onCreateView()" podemos observar que de serie nos da la posibilidad de utilizar un LayoutInflater, un ViewGroup y un Bundle. El LayoutInflater normalmente lo utilizaremos para inflar el layout de nuestro fragment. El ViewGroup será la vista padre donde se insertara el layout de nuestro fragment. Y por último el Bundle podremos utilizarlo para recuperar datos de una instancia anterior de nuestro fragment.

De esta manera ya tendremos creado un fragment que nos devolverá una vista y que podremos insertar en cualquier activity de nuestro proyecto.

El xml del fragmento1 podría ser:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="250dp"
    android:background="#CC00CC00">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Información fragment1" />
</LinearLayout>
```

AGREGAR UN FRAGMENT A UNA ACTIVITY

A la hora de agregar un fragmento a una actividad lo podremos realizar de dos maneras:

1. Declarar el fragmento en el layout de la activity. Este *fragment* tendrá la cualidad de no ser eliminado o sustituido por nada, de lo contrario tendremos errores. Se le da el nombre de **fragment estático o final.**

2. Agregar directamente el Fragment mediante programación Android. Se asocia a un *ViewGroup* (se recomienda el uso de *FrameLayout*). Éste sí que se podrá eliminar o sustituir por otro *fragment* u otro contenido. Se les da el nombre de **fragment dinámico.**

Agregar Fragment mediante layout

Lo primero que tenemos que hacer es crear el layout de nuestra activity especificando un elemento fragment: **Activity_my.xml**

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context=".MyActivity"
    android:id="@+id/activity_my_contenedor">

    <fragment
        android:id="@+id/fragment1"
        android:name="com.xusa.examen.probarfragmentstatico.Fragmento1"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_weight="0.5" />

    <fragment
        android:id="@+id/fragment2"
        android:name="com.xusa.examen.probarfragmentstatico.Fragmento2"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:layout_weight="0.5" />
</LinearLayout>
```

Simplemente creamos el elemento "fragment" y especificamos a través del atributo "android:name" la ubicación de nuestro fragmento (nombre de paquete donde está ubicado el fragmento).

Es recomendable crear un identificador único para cada fragmento que nos puede servir para restaurar fragmentos o incluso para realizar transacciones o eliminación de fragmentos.

Una vez creado el layout de la activity simplemente creamos una actividad y le aplicamos el método "setContentView()" indicando la id del layout que acabamos de crear. El resultado puede ser el siguiente:

- Crea una App para probar los fragments estáticos del ejemplo anterior

Agregar Fragment mediante programación

De esta manera podemos agregar un fragment a una activity en cualquier momento. Simplemente indicaremos la id de una vista padre (ViewGroup) donde deberá colocarse el fragment.

Lo que vamos a hacer es agregar un fragment dinámico junto a uno estático.

Lo primero es definir en el layout de la activity un espacio donde poder añadir el fragment, para ello incluimos un LinearLayout con id “fragment_container”.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MyActivity">

    <fragment
        android:name="com.fragments.ejemplo.ejemplofragments_v1.FragmentUno"
        android:id="@+id/fragment_uno"
        android:layout_width="match_parent"
        android:layout_height="250dp"
        tools:layout="@layout/fragment_uno" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/fragment_container"
        android:orientation="vertical"
        android:background="#3355CC00">


    </LinearLayout>

</LinearLayout>
```

Nos creamos una clase para el fragment2, similar a la anterior y un layout para gestionar el aspecto. Pasemos a añadir el fragment por código. Para ello tenemos que utilizar la API

FragmentTransaction y a través de su método "add()" añadiremos el fragmento a la vista padre. Después de añadir el fragment tendremos que terminar la transacción a través del método "commit()".

Vamos a ver el ejemplo y lo explicamos a continuación:

MyActivity

```
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);

  setContentView(R.layout.activity);

  FragmentManager FM = getSupportFragmentManager();
  FragmentTransaction FT = FM.beginTransaction();

  Fragment fragment = new FragmentDos();
  FT.add(R.id.fragment_container, fragment);
  FT.commit();
}
```

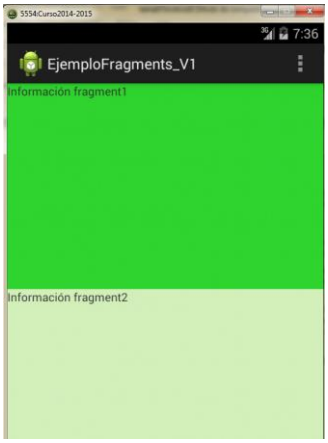
En este caso nuestra activity muestra por defecto un layout "mostrar_fragment_dos.xml" que contiene un ViewGroup, en este caso en un LinearLayout que no muestra nada pero nos va a servir para agregar nuestro fragment en dicha vista padre.

Lo primero que hacemos es crear una instancia de "FragmentManager" a través de su método "getSupportFragmentManaget()". De esta manera estamos creando un objeto que nos servirá para manejar los fragmentos.

A continuación creamos una instancia de "FragmentTransaction" para realizar la transacción de fragmentos. A través del método "beginTransaction()" indicamos que vamos a realizar una transacción de fragmentos.

Lo siguiente es crear una instancia de nuestro "FragmentUNO" y añadirla a la transacción a través del método "add()" que nos pide como parámetros la id del ViewGroup o vista padre donde se colocara dicho fragmento (en este caso es la id del LinearLayout que comentábamos antes) y como segundo parámetro nos pide la instancia del fragmento que se mostrara dicha vista.

Para terminar la transacción siempre deberemos declarar el método "commit()". Y como resultado podremos ver lo siguiente:



- Prueba el ejemplo anterior “fragmentV1”, el fragment1 será estático mientras que el frgament2 será añadido de forma dinámica desde el código.

GESTIONAR FRAGMENTS

En el punto anterior hemos hablado de transacciones, una transacción simplemente es una acción que nos permite agregar, reemplazar, eliminar o incluso realizar otras acciones cuando trabajamos con fragmentos. Estas transacciones pueden ser apiladas por la activity de acogida, permitiendo así al usuario navegar entre fragmentos mientras la activity siga en ejecución.

Cada transacción es un conjunto de cambios que se realizan al mismo tiempo. Podremos realizar dichos cambios a través de los métodos "add()", "replace()", "remove()" terminando la transacción con el método "commit()".

Para añadir la transacción a la pila de retroceso de la activity utilizaremos el método "addToBackStack()" para cada transacción que realicemos. Esta pila será administrada por la activity y permitirá al usuario volver a un fragmento anterior pulsando la tecla volver del smartphone o tablet.

Por otra parte a través del método "setTransaction()" podemos establecer el tipo de animación para cada transacción.

Para este caso se ha creado un ejemplo que muestra un layout con un ViewGroup para mostrar los fragmentos y un botón que servirá para añadir fragmentos a la pila de retroceso. Empezaremos creando el layout que mostrara la activity:

Reemplazar_fragmentos.xml

```
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent">

   <RelativeLayout
      android:id="@+id/fragment_container"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:layout_above="@+id/boton"
      android:layout_below="@+id/texto"/>

   <Button
      android:id="@+id/boton"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:layout_alignParentBottom="true"
      android:text="CAMBIA FRAGMENT"/>
</RelativeLayout>
```

Una vez creado el layout deberemos crear la siguiente activity:

Myactivity.class

```
private boolean bol = false;
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.reemplazar_fragments);

   final Fragment fragmentUNO = new FragmentUNO();
   final Fragment fragmentDOS = new FragmentDOS();

   Button boton = (Button) findViewById(R.id.boton);
   boton.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
         FragmentTransaction FT = getSupportFragmentManager().beginTransaction();
         if (bol) {  FT.replace(R.id.fragment_container, fragmentUNO);}
         else {    FT.replace(R.id.fragment_container, fragmentDOS);   }
         FT.addToBackStack(null);
         FT.commit();
         bol = (bol) ? false : true;
      }
   });
}
```

Comentar que previamente se han creado dos fragmentos "FragmentUNO" y "FragmentDOS", cada uno con su layout.

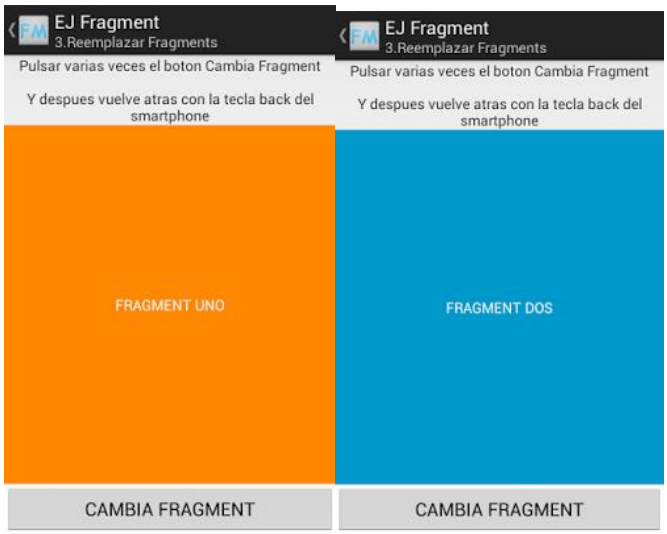
En la activity primero creamos un valor booleano que nos servirá para cambiar entre un fragmento u otro cada vez que el usuario pulse el botón.

Establecemos el layout de la activity y creamos dos instancias de nuestros fragmentos. A continuación declaramos nuestro botón y le aplicamos un listener.

Cada vez que el usuario pulse el botón se creara una transacción añadiendo a la pila un fragmento u otro a través del método "addToBackStack()" que pide como parámetro un tag o identificador para la transacción que se va a realizar.

Se finaliza la transacción con el método "commit()" y cambiamos el valor booleano.

El resultado puede ser el siguiente:



Cada vez que el usuario pulse el botón, se creara una nueva transacción que se irá acumulando en la pila de retroceso. Simplemente tocando la tecla volver de nuestro smartphone o tablet iremos retrocediendo los fragments.

- Prueba el anterior ejemplo FragmentV2: botón que te vaya cargando uno u otro fagment y al mismo tiempo que se acumulen en la pila. Probar que al pulsar el botón de retroceso, se pasa por cada uno de los fragments que se han cargado anteriormente.

COMUNICAR FRAGMENTS Y ACTIVITIES

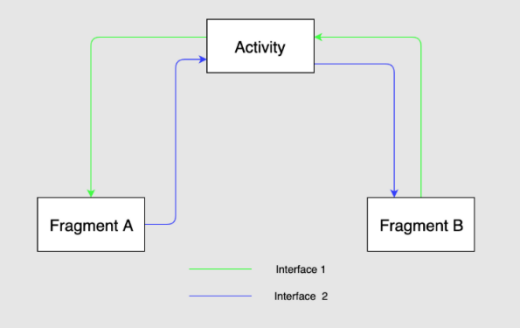
Para comunicar un fragmento con su activity de acogida podemos utilizar el método "getActivity()", por lo tanto podremos localizar una vista de la activity de la siguiente manera:

```
View vista = getActivity().findViewById(R.id.lista);
```

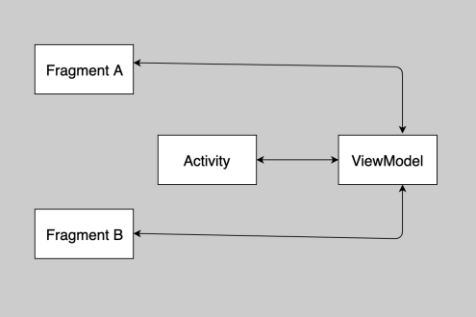
Cuando hablamos sobre la comunicación de fragmentos, debemos tener en cuenta las siguientes premisas:

- Los fragmentos no pueden ni deben comunicarse directamente.
- La comunicación entre fragmentos debe hacerse a través de los asociados activity.
- Los fragmentos no necesitan conocer quién es su actividad principal

A partir de lo anterior, podemos deducir que dos objetos Fragment nunca deben comunicarse directamente. Hay varias maneras de realizar la comunicación entre estos, pero vamos a estudiar las dos más recomendadas. Estas dos formas son: mediante la implementación de una interface o mediante un elemento ViewModel.



Comunicación mediante Interface



Comunicación mediante ViewMode

Comunicación mediante ViewModel

ViewModel es una de las nuevas incorporaciones a la implementación de Android y la que google recomienda para en la comunicación entre fragments.

Un ViewModel siempre se crea en asociación con un ámbito (un fragmento o una actividad) y se conservará mientras el ámbito esté activo. Por ejemplo, si es una Actividad, hasta que se termine. En otras palabras, esto significa que un ViewModel no se destruirá si su propietario se destruye por un cambio de configuración (por ejemplo, rotación). La nueva instancia del propietario se volverá a conectar al ViewModel existente.

Para poder crear un ViewModel, deberemos en primer lugar añadir la siguiente dependencia:

```
'androidx.lifecycle: lifecycle-extensions:2.1.0'
```

El siguiente paso será crear una clase que derive de ViewModel parecida a la siguiente, y teniendo en cuenta que tipo de datos queremos pasar entre los fragments:

```
public class ViewModelFragment extends ViewModel {

        private MutableLiveData<String> liveData;
        public LiveData<String> getData() {
            if (liveData != null)  setData(liveData.getValue());
            else liveData= new MutableLiveData<String>();
            return liveData;
        }
        public void setData(String t) {
            if (liveData == null)  liveData= new MutableLiveData<String>();
            liveData.setValue(t);
        }
}
```

Como podemos ver en el código anterior, aparece un elemento nuevo para exponer los datos llamado LiveData, este elemento es un titular de datos que es capaz de ser observado para enviar solo actualizaciones de datos cuando su observador está activo, puede contener

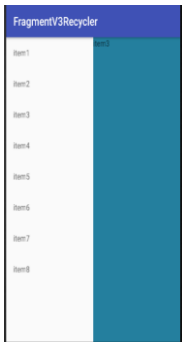
cualquier tipo de datos, y además de eso, es consciente del ciclo de vida para mandar las actualizaciones de datos solamente si el observador está activo.

Para observar un LiveData, hay dos maneras de hacerlo mediante observe ***(LifecycleOwner, Observer)*** u ***observeForever (Observer)***. La forma más estándar es:

observe (LifecycleOwner, Observer): esta forma vincula al observador con un ciclo de vida, cambiando el estado activo e inactivo de LiveData según el estado actual de LifecycleOwner

Vamos a suponer un ejemplo en el que en una activity se cargan dos fragments a la vez. Uno tendrá una lista y en el otro se mostrará el elemento pulsado de la lista.

Donde la actividad para cargar los fragments, podría ser así.




```
public class MyActivity2 extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_my2);
        FragmentManager FM = getSupportFragmentManager();
        FragmentTransaction FT = FM.beginTransaction();
        Fragment fragmentLista = new MyListFragment();
        Fragment fragmentSecundario = new FragmentSecundario2();
        FT.replace(R.id.fragment_container2, fragmentSecundario);
        FT.replace(R.id.fragment_container1, fragmentLista);
        FT.commit();
    }
}
```

El código del fragment con la lista sería algo como lo que sigue:

```
public class MyListFragment extends Fragment{
    RecyclerView recyclerView;
    Adapter adapter;
    ArrayList<String> valores;
    ViewModelFragment pageViewModel;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        valores=new ArrayList<>();
        pageViewModel = ViewModelProviders.of(getActivity()).get(ViewModelFragment.class);
        rellenarValores();
    }
    private void rellenarValores() {
        String[] array= new String[]{"item1", "item2", "item3", "item4", "item5", "item6", "item7", "item8"};
        for(String x:array)valores.add(x);
    }
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        super.onCreateView(inflater, container, savedInstanceState);
        View rootView = inflater.inflate(R.layout.fragment_main, container, false);
        recyclerView=(RecyclerView)rootView.findViewById(R.id.recyclerX);
        recyclerView.setHasFixedSize(true);
        adapter=new Adapter(valores);
        recyclerView.setAdapter(adapter);
        recyclerView.setLayoutManager(new LinearLayoutManager(getActivity(), LinearLayoutManager.VERTICAL, false));
        adapter.miOnClick((v) -> {
                pageViewModel.setData(valores.get(recyclerView.getChildAdapterPosition(v)));
        });
         return rootView;
    }
}
```

Con la siguiente línea, lo que hacemos es inicializar nuestro objeto ViewModel, esto tendremos que hacerlo en las clases que vayan a comunicarse.

```
pageViewModel = ViewModelProviders.of(getActivity()).get(ViewModelFragment.class);
```

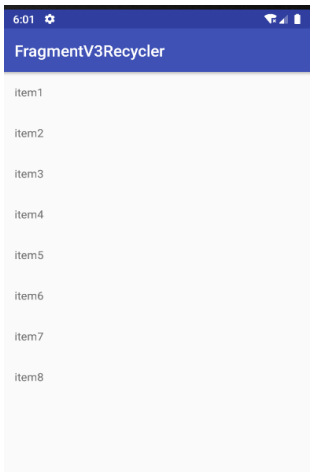

Con la otra línea subrayada en el código, se puede ver que lo que hace es asignar la información que queremos comunicar. Para ello usamos el método setData que hemos implementado en nuestra clase. Si nos remitimos al código de la clase ViewModelFragment, veremos que en el método setData lo que hacemos es asignar el dato recibido al objeto de tipo LiveData.

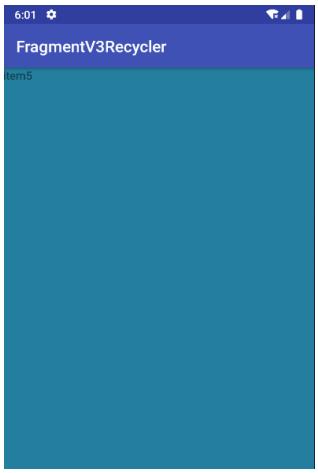
En el otro fragmento, tenemos el código siguiente:

```
public class FragmentSecundario2 extends Fragment {

    ViewModelFragment pageViewModel;
    TextView textView;
    @Override
    public void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); }

    @Override
    public View onCreateView(LayoutInflater inflater,ViewGroup container,Bundle saveInstanceState){
        View rootView = inflater.inflate(R.layout.fragment_secundario, container, false);
        pageViewModel = ViewModelProviders.of(getActivity()).get(ViewModelFragment.class);
        textView=((TextView) rootView.findViewById(R.id.texto));
        pageViewModel.getData().observe(this, new Observer<String>() {
            @Override
            public void onChanged(String s) {
                textView.setText(s);
            }
        });
        return rootView;

    }
}
```

Por un lado, inicializamos el objeto y luego lo pondremos a observar por si ocurre un cambio en el estado del dato (LiveData), de forma que cuando se produce el cambio pasa a modificarse el texto del fragment con el dato de entrada.

Otro caso distinto, podemos tenerlo cuando el fragment se comunica a través de la activity, mediante el ViewModel. Vamos a suponer, el caso más común en el que una activity cargará una lista y al pulsar un elemento de esta, se carga el otro fragmento.





En este caso, el código de la actividad principal sería:

```
public class MyActivity extends AppCompatActivity {
    ViewModelFragment pageViewModel;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        pageViewModel = ViewModelProviders.of(this).get(ViewModelFragment.class);
        setContentView(R.layout.activity_my);

        FragmentManager FM = getSupportFragmentManager();
        FragmentTransaction FT = FM.beginTransaction();
        Fragment fragmentLista = new MyListFragment();
        FT.replace(R.id.fragment_container, fragmentLista);
        FT.commit();
         pageViewModel.getData().observe(this, new Observer<String>() {
            @Override
            public void onChanged(String s) {
                Fragment fragmentSecundario = new FragmentSecundario();
                Bundle args = new Bundle();
                args.putString("str", s);
                fragmentSecundario.setArguments(args);
                FragmentManager FM = getSupportFragmentManager();
                FragmentTransaction FT = FM.beginTransaction();
                FT.replace(R.id.fragment_container, fragmentSecundario);
                FT.addToBackStack(null);
                FT.commit();
            }
        });
    }
}
```

Donde podemos ver la inicialización del objeto ViewModel y por otra parta la escucha por si se produce algún cambio en el LiveData. En el método de observación, aprovechamos para lanzar el fragmento secundario pasándole la información mediante un bundle, como ya hemos hecho en otros casos.

El código del FragmentSecundario podría ser como el siguiente:

```
public class FragmentSecundario extends Fragment {
    TextView textView;
    @Override
    public void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); }

    @Override
    public View onCreateView(LayoutInflater inflater,ViewGroup container,Bundle saveInstanceState){
        View rootView = inflater.inflate(R.layout.fragment_secundario, container, false);
        textView=((TextView) rootView.findViewById(R.id.texto));
        textView.setText(getArguments().getString("str"));
        return rootView;
    }
}
```

Solamente se recupera el Bundle mediante getArguments().getString(), para recibir la información que le mandó la actividad principal.

Si no puedes utilizar un ViewModel compartido para la comunicación entre objetos Fragment, puedes implementar un flujo de comunicación manualmente mediante interfaces. Sin embargo, esto da como resultado un mayor esfuerzo de implementación y no puede reutilizarse en otros objetos Fragment con facilidad.

Comunicación mediante Interface

La otra manera correcta para pasar información y controlar algún evento, es crear una interface en el fragmento y exigir a la activity que la implemente. De esta manera cuando el fragmento reciba un evento también lo hará la activity, que se encargara de recibir los datos de ese evento y compartirlos con otros fragmentos.

Para ello vamos a realizar una aplicación que contenga un listView y cuando pulsemos sobre un elemento de la lista nos visualice en otro fragment qué elemento hemos pulsado.

El xml de la activity principal contendrá un layout con nombre "fragment_container", donde cargaremos en primer lugar la lista (en nuestro caso un ListFragmnet) y posteriormente al pulsar un elemento de la misma, el otro fragment. Empecemos definiendo y declarando la interface:

```
public interface OnSelectedItemListener {
   public void onItemSelected(String str);
}
```

A esta interface la hemos llamado "onSelectedItemListener" y simplemente contiene un método llamado "onItemSelected" con un único parámetro.

Veamos el código java del fragment que contiene la lista. Posteriormente añadiremos un objeto de este tipo en la activity principal.

```
public class MyListFragment extends ListFragment{
    private String[] valores={"item1","item2","item3","item4","item5","item6","item7","item8"};
    private OnSelectedItemListener listener;

    @Override
    public void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);

        setListAdapter(new ArrayAdapter(getActivity(),android.R.layout.simple_list_item_1,valores));
    }

    @Override
    public void onListItemClick(ListView l,View v,int position,long id){
        super.onListItemClick(l,v,position,id);
        listener.onSelectedItem(valores[position]);
    }

    @Override
    public void onAttach(Context context) {
        super.onAttach(context);
        try{
            listener=(OnSelectedItemListener) context;
        } catch (ClassCastException e){}
    }
```

Sobreescribimos el método "onListItemClick" (pulsación sobre un elemento de la lista) y que envía el evento y los datos del evento a la interface. De esta manera cada vez que el fragmento reciba un evento, automáticamente lo mandara a la interface que a su vez lo recibirá la activity:

Para asegurar que la Activity de acogida implementa la interface vamos a sobreescribir el método "onAttach()" que será llamado cada vez que la activity cree una instancia del fragmento.

```
@Override
public void onAttach(Context context) {
    super.onAttach(context);
    try{
        listener=(OnSelectedItemListener) context;
    } catch (ClassCastException e){}
}
```

Para terminar deberemos crear la activity principal e implementar la interface:

```
package com.example.josebalmasedadelalamo.ejemplofragments_v3;

import ...

public class MyActivity extends FragmentActivity implements OnSelectedItemListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_my);

        FragmentManager FM=getSupportFragmentManager();
        FragmentTransaction FT=FM.beginTransaction();

        Fragment fragmentLista=new MyListFragment();
        FT.replace(R.id.fragment_container,fragmentLista);
        FT.commit();
    }

    @Override
    public void onSelectedItem(String str) {
        Fragment fragmentSecundario=new FragmentSecundario();

        Bundle args=new Bundle();
        args.putString("str",str);
        fragmentSecundario.setArguments(args);

        FragmentManager FM=getSupportFragmentManager();
        FragmentTransaction FT=FM.beginTransaction();

        FT.replace(R.id.fragment_container,fragmentSecundario);
        FT.addToBackStack(null);
        FT.commit();
    }
```

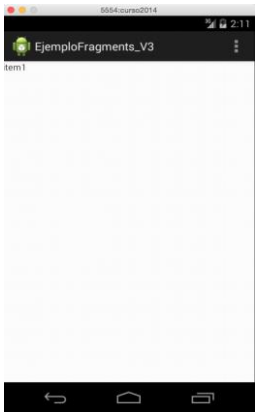
Lo que hacemos es añadir de forma dinámica el fragment de la lista a la vista principal. Posteriormente, implementamos la interface.

En el método que nos obliga a sobreescribir la interface guardamos los datos que recibimos de cada evento del fragment en un Bundle y se lo aplicamos a un nuevo fragment. A continuación se crea una transacción a este nuevo fragment y se añade a la pila de retroceso.

Para terminar, el nuevo fragment recuperara los datos de la siguiente manera:

Es importante tener en cuenta los xml necesarios: uno para la activity principal y otro para el fragment secundarios, el fagment que contiene la lista no necesita xml porque hemos implementado un ListFragment, si fuera un fragment normal, tendríamos que definir un xml también para él.

```java
package com.example.josebalmasedadelalamo.ejemplofragments_v3;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

public class FragmentSecundario extends Fragment {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }

    @Override
    public View onCreateView(LayoutInflater inflater,ViewGroup container,Bundle saveInstanceState){
        View rootView = inflater.inflate(R.layout.fragment_secundario, container, false);

        Bundle b=getArguments();
        String inf= (String) b.get("str");
        ((TextView) rootView.findViewById(R.id.texto)).setText(inf);

        return rootView;

    }
}
```



- Crea la app FragmentV3 que permita probar el ejemplo anterior con una ListView

Fragment con RecyclerView

En el caso de querer aplicar el concepto anterior a recyclerView los cambios serían mínimos. Tendríamos que crear un layout para que contuviera el RecyclerView:

```xml
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
    android:layout_height="match_parent" android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context=".MainActivity$PlaceholderFragment">
    <android.support.v7.widget.RecyclerView android:id="@+id/recyclerX"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
    </android.support.v7.widget.RecyclerView>
</RelativeLayout>
```


Otro para la línea del RecyclerView:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView android:id="@+id/linea_lista" android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

El Adaptado y el holder como lo hemos hecho siempre, en el adaptador implementaremos el OnClickListener para que detecte la pulsación sobre un elemento de la lista que nos permitirá cargar el segundo Fragment.

La clase Fragment que carga el recycler, tendrá que derivar de Fragment e implementar el método onCreateView de la siguiente manera:

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
    super.onCreateView(inflater, container, savedInstanceState);
    View rootView = inflater.inflate(R.layout.fragment_main, container, false);
    recyclerView=(RecyclerView)rootView.findViewById(R.id.recyclerX);
    recyclerView.setAdapter(adapter);
    recyclerView.setHasFixedSize(true);
    adapter=new Adapter(valores);
    recyclerView.setLayoutManager(new LinearLayoutManager(getActivity(), LinearLayoutManager.VERTICAL, false));
    adapter.miOnClick(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            listener.onSelectedItem(valores.get(recyclerView.getChildAdapterPosition(v)));
        }
    });

     return rootView;
}
@Override
public void onAttach(Context context) {
    super.onAttach(context);
    try{
        listener=(OnSelectedItemListener) context;
    } catch (ClassCastException e){}
}
```

El resto de código no variará del implementado para la lista.

- **Ahora crea la app FragmentV3Recycler que haga lo mismo que en el caso anterior pero con un RecyclerView**
- **EjercicioResueltoAves con RecyclerView.**

DIALOGOS FRAGMENTS

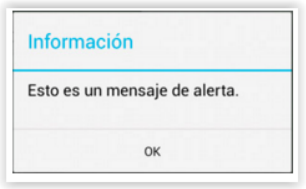
Como hemos visto al principio del tema, la clase Fragment tiene varias derivadas que nos permiten trabajar con ellas para realizar tareas concretas. Para terminar este tema, veremos los Dialogos derivados de la clase DialogFragment. Para utilizar este tipo de diálogos deberemos crearnos una clase derivada de DialogFragment e implementar en ella el tipo de dialogo que queramos, usando un objeto de la clase AlertDialog (como esta parte ya la conocemos de temas anteriores, solo vamos a comentar un par de AlertDialog diferentes). El paso diferente al de los diálogos sin fragments, es que en vez de

mostrarlos directamente con show() tendremos que cargar el fragment mediante un FragmentManager. Vamos a pasar a ver un ejemplo:

Diálogo de Alerta

Este tipo de diálogo se limita a mostrar un mensaje sencillo al usuario, y un único botón de OK para confirmar su lectura. Veamos un ejemplo:

```
public class DialogoAlerta extends DialogFragment {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {

        AlertDialog.Builder builder =
                new AlertDialog.Builder(getActivity());

        builder.setMessage("Esto es un mensaje de alerta.")
               .setTitle("Información")
               .setPositiveButton("OK", new DialogInterface.OnClickListener() {
                   public void onClick(DialogInterface dialog, int id) {
                       dialog.cancel();
                   }
               });

        return builder.create();
    }
}
```



Para lanzar este diálogo por ejemplo desde nuestra actividad principal, obtendríamos una referencia al Fragment Manager mediante una llamada a getSupportFragmentManager(), creamos un nuevo objeto de tipo DialogoAlerta y por último mostramos el diálogo mediante el método show() pasándole la referencia al fragment manager y una etiqueta identificativa del diálogo.

```
btnAlerta.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        FragmentManager fragmentManager = getSupportFragmentManager();
           DialogoAlerta dialogo = new DialogoAlerta();
        dialogo.show(fragmentManager, "tagAlerta");
    }
});
```

Si por algún motivo se necesita pasar información al dialogo, podremos usar un método estático al que llamaremos para crear el objeto, y al que le pasaremos los elementos necesarios. En ese método crearemos un Bundle para añadirlo al nuevo objeto devuelto y que luego será recuperado en el método onCreateDialog.

```
public class DialogoAlerta extends DialogFragment {

    static DialogoAlerta crearNuevoDialogo(String cadena)
    {
        Bundle args = new Bundle();
        args.putString("CADENA",cadena);
        DialogoAlerta dialogo=new DialogoAlerta();
        dialogo.setArguments(args);
        return  dialogo;
    }
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        super.onCreateDialog(savedInstanceState);
        AlertDialog.Builder builder=new AlertDialog.Builder(getActivity());
        final String cadena=getArguments().getString("CADENA");
        builder.setMessage(cadena)
                .setTitle("Información")
                .setPositiveButton("OK", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialogInterface, int i) {
                        dialogInterface.cancel();
                    }
                });
        return builder.create();
    }
}
```


La llamada al Dialogo:

```
DialogoAlerta.crearNuevoDialogo("ESTO ES UN DIALOGO DE ALERTA").show(getSupportFragmentManager(),"Alerta");
```

Diálogo de Selección Múltiple

Este tipo de dialogo nos permite implementar una selección múltiple, como podemos ver en la siguiente imagen:

```
builder.setTitle("Selección")
    .setMultiChoiceItems(items, null,
        new DialogInterface.OnMultiChoiceClickListener() {
        public void onClick(DialogInterface dialog, int item, boolean isChecked) {
             Log.i("Dialogos", "Opción elegida: " + items[item]);
        }
});
```

Igual que en los diálogos que no derivan de Fragment, podremos crear nuestros propios diálogos personalizados.

- **Añade el Click largo en el EjercicioResueltoAves de forma que al realizar una pulsación larga sobre uno de los elementos del recycler, se mostrará un FragmentDialog que nos pregunte si queremos eliminar un elemento, y que pase a eliminarlo en caso afirmativo.**
- **Ejercicio propuesto Agenda.**